PARSEC

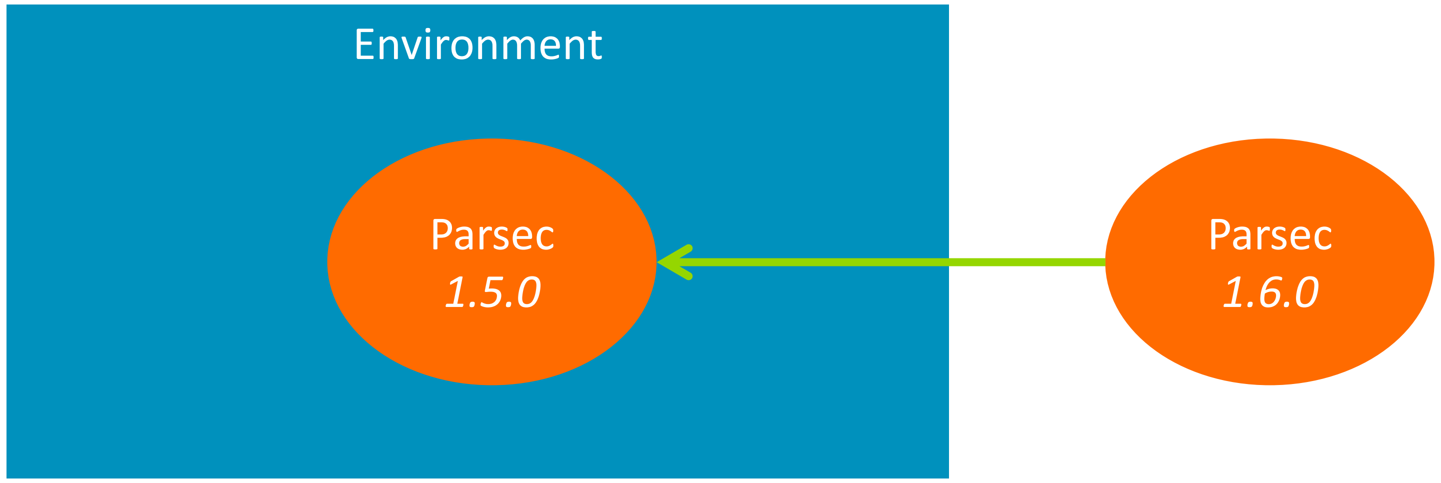
Making Parsec Stable

Hugues de Valon

May 2021

Our definition of stability

- *"whatever two versions A and B of the Parsec binary, B being newer than A, A and B are stable if A can be removed and replaced by B without breaking anything."*
- Rollbacks not supported*
- "without breaking anything" -> does not completely fail into an unusable state
- Semantic versioning
- Different stability requirements (10)



Stability requirements

1. Communication with clients
    1. Definition of requests/responses (their format)
    2. Definition of operation contracts (their intended behaviour)
    3. Definition of Listeners endpoints (how to contact Parsec)
2. Communication with authenticators
3. Communication received from the CLI invocation
4. Configuration file (including default configuration options)
5. Persistent data stored internally (only key mappings currently)
6. OS signals
7. systemd communication
8. Dynamic libraries dependencies
9. Environment variables
10. Toolchain

PARSEC

Stability requirements, the good

1. Communication with clients
   1. Definition of requests/responses (their format)
   2. Definition of operation contracts (their intended behaviour)
   3. Definition of Listeners endpoints (how to contact Parsec)
2. Communication with authenticators
3. Communication received from the CLI invocation
4. Configuration file (including default configuration options)
5. Persistent data stored internally (only key mappings currently)
6. OS signals
7. systemd communication
8. Dynamic libraries dependencies
9. Environment variables
10. Toolchain

PARSEC

Stability requirements, the good

- Listing precise items to be checked, complete the page in the book
- Separate code into their own files and set up code owners
- Add warnings in the code as comment for stability and incentive to write new tests for new requirements that appear
- Add specific tests for each of the requirements, executed on each Parsec PR
- Old stability tests should not be removed!

- For communication with clients: tests needed on each clients’ version as well to ensure it works for different versions.

PARSEC

Stability requirements, the bad

1. Communication with clients
   1. Definition of requests/responses (their format)
   2. Definition of operation contracts (their intended behaviour)
   3. Definition of Listeners endpoints (how to contact Parsec)
2. Communication with authenticators
3. Communication received from the CLI invocation
4. Configuration file (including default configuration options)
5. Persistent data stored internally (only key mappings currently)
6. OS signals
7. systemd communication
8. Dynamic libraries dependencies
9. Environment variables
10. Toolchain

PARSEC

Stability requirements, the bad

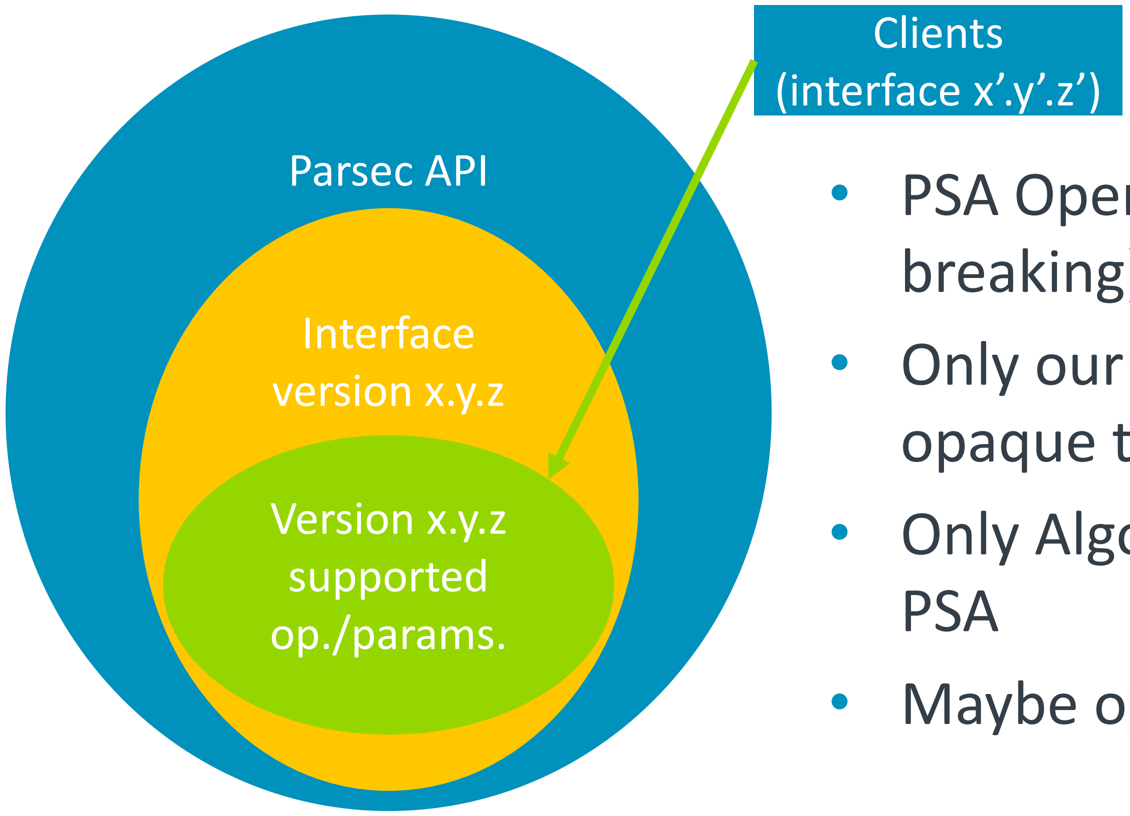
Definition of requests/responses (their format)

- Contracts respect open/close principle so we are good in principle
- But what do we do with PSA Crypto upgrades???
- Sticking with 1.0.0 is a big maintenance burden and can lead to complicated handling
- Allowing non-breaking change is easier but more operations are needed
- Deserializing new data with old interface:
  - New fields will be ignored. Must be optional.
  - New enum variants (enum and oneof) will be set to None.
    - In a request deserialized by the service: PsaErrorNotSupported returned + capabilities check (next slide)
    - In a response deserialized by the client: PsaErrorNotSupported only (+ logs about what is not recognized). Currently only ListKeys, could be best-effort instead of fail-all.

Stability requirements, the bad

Definition of requests/responses (their format)

- Clients want to know “what is the service interface” to avoid PsaErrorNotSupported
- But actually more “what are the supported parameters of supported operations”



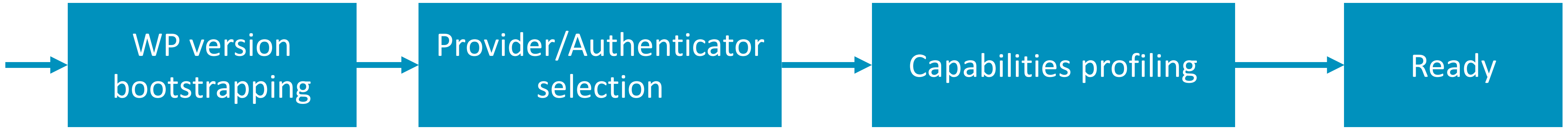
- PSA Operations can not have new fields (otherwise breaking)
- Only our conversions: integer -> enums/oneoff and opaque types -> defined types
- Only Algorithm and Key attribute types (recursively) for PSA
- Maybe our own types in the future

PARSEC

Stability requirements, the bad

Definition of requests/responses (their format)

- GetSupportedAlgorithms (#389)
- GetSupportedKeyAttributes for generation or import
- Would not need to request it often (until the service is reloaded), could be stored (raw or analysed)
- Helps for stability and opens the door to profiling (Higher level client!!!)



Ping

ListProviders
ListAuthenticators

ListOpcodes
GetSupportedAlgorithms
GetSupportedKeyAttributes

All op. send to the service should succeed!

Stability requirements, the bad

1. Communication with clients
    1. Definition of requests/responses (their format)
    2. Definition of operation contracts (their intended behaviour)
    3. Definition of Listeners endpoints (how to contact Parsec)
2. Communication with authenticators
3. Communication received from the CLI invocation
4. Configuration file (including default configuration options)
5. Persistent data stored internally (only key mappings currently)
6. OS signals
7. systemd communication
8. Dynamic libraries dependencies
9. Environment variables
10. Toolchain

Stability requirements, the bad

Persistent data stored internally (only key mappings currently)

- Current KeyInfoManager is not the most idiomatic (using directory structure), hard to modify
- UUID to be used instead of Provider ID (or “name” + UUID if dynamic #404)
- Authenticator namespace to be added to support multiple ones
- Protobuf’s version of the Attributes for stability (because of previous slides)
- Add a new KeyInfoManager based on SQLite

Name	Type	Properties
Authenticator	INTEGER	PRIMARY KEY
Application Name	TEXT	PRIMARY KEY
UUID	TEXT	PRIMARY KEY
Provider Name	TEXT	PRIMARY KEY
Key Name	TEXT	PRIMARY KEY
Key ID	BLOB	
Key Attributes	BLOB	

Steps forward

- Agreeing on the changes
    1. Allowing non-breaking changes
    2. New discovery operations
    3. SQLite-based KIM
    4. Allowing multiple authenticators
    5. Provider name as a feature
- Start implementing! Aim to have everything for 0.8.0
- If it works, next version will be 1.0.0

PARSEC